Automatic Generation of Controllers for Collision-Free Flexible Manufacturing Systems

Mohammad Reza Shoaei, Bengt Lennartson and Sajed Miremadi
Department of Signals and Systems, Chalmers University of Technology
SE-412 96, Gothenburg, Sweden
{shoaei, bengt.lennartson, miremads}@chalmers.se

***Abstract*—A method for automatic generation of non-blocking controllers that generate collision-free flexible manufacturing cells is presented in this paper. Today, industry demands on flexible production sometimes require significant changes in location, orientation and configuration of industrial robots and other moving devices, when new products are introduced. All these changes pose a threat to the devices to collide while sharing workspace. Although the use of simulation software to facilitate these changes is gaining popularity, the coordination of collision-free flexible manufacturing systems is still at best a semi-manual trial-and-error procedure. To avoid this, a formal model of the operations in a manufacturing system is generated, and for each operation state a corresponding 3D simulation shape is created. A collision-free system is then achieved by considering pairs of colliding shapes as forbidden states. The automatic generation also includes a synthesis procedure, where a non-blocking and controllable supervisor is generated based on guard generation. The guards are computed by binary decision diagrams, which means that complex systems can be handled, still generating comprehensible restrictions that are easily included in PLC-code.**

I. Introduction

Today's automotive industry trend towards accelerated product development cycles, and the ambition to shorten the time-to-market, represent the symptoms of an extremely competitive marketplace. This has driven the industry to develop and coordinate highly complex *flexible manufacturing systems (FMS)*. A key enabler to handle this type of complexity is to replace time-consuming on-line and manual tests with virtual development, including off-line programming and up-front performance simulation.

Most digital manufacturing software, such as [1], [2], and [3], offer tools to facilitate the development of control code, by providing the functionality for off-line programming, digital manufacturing simulation and virtual commissioning. Still, the coordination of FMS is at best a semi-manual trial-and-error procedure, where the generation of control code is often a time demanding occupation, suffering from human mistakes and risks for collisions in the final execution of the controlled FMS.

The option to use formal methods, including controller synthesis to automatically generate correct control functions, is still very limited in industry. Some reasons are given in [4], including modeling difficulties and state space explosion. This problem is further discussed in [5], [6], and [7], where in the latter a formal language for hierarchical operations and sequences of operations (SOPs) are introduced to support controller synthesis.

In synthesis procedures, such as *supervisory control theory* (SCT), introduced by Radmadge and Wonham [8], the generation of the supervisor depends on a specification of the closed loop behavior. For FMSs some parts of this specification result from production planning, where engineers naturally introduce desired operations and SOPs, for instance by the language introduced in [7]. Other parts of the specification are based on safety issues, more related to the coordination between robots and other moving devices, such as machines, conveyors, fixtures, and clamps. These safety specifications depend on the physical layout of the FMS, and actual geometry of the parts to be manipulated. Hence, rapid changes in an FMS may imply large changes of these safety specifications.

To handle this challenging problem, an automatic generation of safety specifications is presented in this paper, to avoid collisions between any moving devices in an FMS. This is achieved by first identifying all areas where robots and other devices are performing operations in a shared workspace. To avoid collisions, a set of volumes (henceforth called *shapes*), which represents the location of devices in the workspace, are created for all involved operations, based on simulations in a 3D simulation environment. Pairwise intersections of theses shapes are identified, and avoided by adding guards to the corresponding operation models, to eliminate all possible collisions.

A consequence of introducing these guards, as well as the user specifications in terms of SOPs mentioned above, blocking and uncontrollability problems may arise. These problems are avoided by synthesizing a supervisor that prevent some controllable events to be executed, to fulfill the desired specification [8]. Since the operations and SOPs are modeled by a new type of automata with variables called Extended Finite Automata (EFA) [9], the synthesis is also based on EFA. Applying a recently developed strategy [10], additional guards are then added to achieve the required closed loop behavior. The computations, including the guard generation, are based on *binary decision diagrams* (BDDs) [11], which means that large and complex systems can be handled efficiently, where small and comprehensible guards often can be achieved that correspond to a large number of states in a traditional supervisor implementation. This is exemplified in a case study in the end of this paper.

The ambition of this work is to be easily applicable, based on currently available software. For this purpose, functionality that is considered to be a part of most digital manufacturing software is used as far as possible. For demonstration purposes, the method presented in this paper is implemented using *Dassault Systèmes DELMIA V5* [1], a software that easily generates the required pairwise intersected shapes, and *Supremica* [12], a tool for formal verification and synthesis developed by the Chalmers University of Technology.

This paper is organized as follows, in Section II mathematical preliminaries are shortly reviewed. Section III is devoted to a presentation of the method in detail, followed by Section IV, where a case study is given. Conclusions and future work are presented in Section V.

II. Preliminaries

For notational convenience, the moving devices that are employed in a cell, such as robots, machines, conveyors, fixtures, clamps etc, are called *resources*. Each resource in a cell is assumed to be assigned a number of *operations* to perform, like *grip* or *weld* for a robot and *close* or *open* for a clamp.

A. Extended Finite Automata

In order to describe the behaviour of a cell in a structured way, *extended finite automata* [9] are used. An EFA is defined as a 7-tuple $E = \langle Q \times V, \Sigma, \mathcal{G}, \mathcal{A}, \rightarrow, (q_0, v_0), M \rangle$. The set $Q \times V$ is the extended finite set of states, where Q is a finite set of *discrete locations* and V is the finite domain of an m-tuple of variables, $v = (v^1, v^2, \ldots, v^m)$. Σ is a nonempty finite set of events (the alphabet). $\mathcal{G}$ is a set of guard predicates over V, $\mathcal{A}$ is a set of action functions from V to V, where each function maps the present variable values to the variable values of the next state. $\rightarrow \subseteq Q \times \Sigma \times \mathcal{G} \times \mathcal{A} \times Q$ is a state transition relation, $(q_0, v_0) \in Q \times V$ is the initial state, and $M \in Q \times V$ is a set of marked (desired) states.

The event set, is partitioned into two disjoint subsets, the *controllable* events Σ_c and the *uncontrollable* events Σ_{uc}. A controllable event can be inhibited by the supervisor, while an uncontrollable event cannot be inhibited. When an uncontrollable event occurs in the plant, the supervisor must be able to follow it, or risk losing control over the system.

EFAs are composed by *full synchronous composition* [13]. In the composition of two EFAs, a shared event is enabled if and only if it is enabled by each of the composed automata [9]. There are three types of automata appearing in this paper, *plants, specifications*, and *supervisors*. The plant is a model of all operations in the system, and the specification is a model of the desired behavior of the controlled system. The supervisor is *synthesized* from the plant and the specification, using the *supervisory control theory* (SCT) [8]. It issues control over the plant so that the specification is fulfilled.

B. Simulation Assumptions

It is assumed that resources are collision free in the initial state of a cell. This is required, since the method otherwise identifies the initial location as a forbidden state and returns the empty controller. Also, it is assumed that the operations are defined and available in the 3D simulation model. Furthermore, the exactness of the shapes is very valuable for this method. Therefore, it is assumed that the created shapes using the software functions are accurate enough. Finally, the cell behavior is analyzed for one cycle, since the workspace for the resources are assumed to be the same for all cycles.

III. Method

The method used to generate a controller that guarantees a collision-free FMS consists of the following main steps

A Operation model generation
B Collision detection
C Interlocks generation
D User specification
E Synthesis

In the first step, operations are modeled by using EFAs. Then the set of shapes from all operations are created in step B, by using Dassault Systèmes DELMIA V5 simulation software and simulating the operations. Collisions are identified by pairwise intersection tests over the set of shapes, and the pairs of colliding shapes are generated. In step C, based on the pairs of colliding shapes, interlocks are added to the corresponding operation models. In the final step, including the given user specification, a non-blocking and controllable supervisor is generated.

A. Operation Model Generation

To be able to formally specify properties and relations of operations with respect to resources in the manufacturing system, the *operation* model in [7] is used.

For notational purposes, a slight reformulation of the EFA is done in the operation model, by replacing the guards and actions by a set C of transition conditions.

An operation is an EFA where the set of discrete locations $Q_k = \{O_k^i, O_k^e, O_k^f\}$, the event set $\Sigma_k = \{O_k^\uparrow, O_k^\downarrow\}$, the set of transition conditions $C_k = \{C_k^\uparrow, C_k^\downarrow\}$, the transition relation $\rightarrow_k = \{\langle O_k^i, O_k^\uparrow / C_k^\uparrow, O_k^e \rangle, \langle O_k^e, O_k^\downarrow / C_k^\downarrow, O_k^f \rangle$, and the initial and marked locations are $q_k^i = O_k^i$ and $q_k^m = O_k^f$, respectively. See Fig. 1.

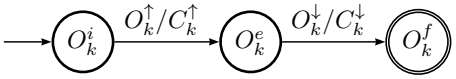

Fig. 1. The EFA model of the operations. The discrete locations O_k^i, O_k^e and O_k^f where $O_k \in \mathcal{O}$ represent the initial, execution and completion of the operation, respectively.

The transition for operation O_k from the initial discrete location O_k^i to the execution location O_k^e is enabled when the *precondition* $C_k^\uparrow$ is satisfied, after which the transition can be fired and the start event $O_k^\uparrow$ occurs. In the same way the completion event $O_k^\downarrow$ can only occur when the *postcondition* $C_k^\downarrow$ is fulfilled. The basic assumption is that all

operations can be executed concurrently in parallel. But the physical location of the resources further reduces the flexibility and typically leads to restrictions on the operations. Therefore, some operations must wait until other operations are finished. These sequential restrictions on the order between different operations are formally expressed by logical preconditions.

In this paper, the event $O_k^{\downarrow}$ is assumed to be an uncontrollable event. The motivation is that in practice, the start of an operation, $O_k^{\uparrow}$, can be disabled by the controller at any time, whereas the completion of the operation, $O_k^{\downarrow}$, is performed autonomously by the system without any interference with the controller.

Example 1 (A small manufacturing cell): Consider a manufacturing cell including three resources, two robots and a clamp, and a number of operations for each resource to perform, e.g. *visit a number of targets* from home position for robots and *open* for the clamp.

The set of resources of the cell is $\mathcal{R} = \{Rb1, Rb2, C\}$ where $Rb1$, $Rb2$ and C are assigned to perform the operations $\{O_1, O_2\}$, $\{O_3, O_4\}$ and $\{O_5\}$, respectively. It is assumed that the operation O_1 is followed by O_2 and operation O_3 is followed by O_4. Hence, to formally express this, a true condition on the discrete locations O_1^f and O_3^f are added to the transition conditions $C_2^{\uparrow}$ and $C_4^{\uparrow}$, see Fig. 2. □

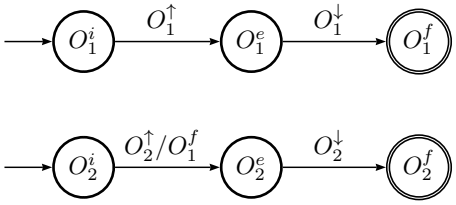

Fig. 2. The EFA models of the operations O_1 and O_2. The sequence O_1 followed by O_2 is achieved by the condition O_1^f on the event $O_2^{\uparrow}$.

B. Collision Detection

In this step, an algorithm for collision detection is presented, see Algorithm 1. Collision detection is a well established research field, see e.g. [14] and [15]. In this work we simply use available 3D simulation software functionality to find actual collisions. The collisions are identified by pairwise intersection test over the set of shapes generated from simulating the operations. The pairs with intersected shapes, excluding shapes from the same resource, will be added to the set of pairs of colliding shapes $\mathcal{X}$.

To map the volume of the resource in the workspace while performing the operation O_k, the set of shapes $\mathcal{S}_k = \{S_k^i, S_k^e, S_k^f\}$ is introduced. The shapes S_k^i, S_k^e and S_k^f are the 3D simulation volumes of the resource performing the operation O_k in the stand still position at the initial location O_k^i, in the sweep mode at the execution location O_k^e, and the stand still position at the finished location O_k^f.

Consider two operations O_{k_1} and O_{k_2} performed in sequence by a resource R. Then the shape $S_{k_1}^f$ of the final location $O_{k_1}^f$ in the predecessor operation O_{k_1}, is identical to the shape $S_{k_2}^i$ of the initial location $O_{k_2}^i$ in the successor operation O_{k_2}. Therefore, it is enough to add one shape called S_{k_1,k_2}^{fi}, which denotes the identical shapes $S_{k_1}^f$ and $S_{k_2}^i$. This shape represents the combination of discrete locations $O_{k_1}^f$ and $O_{k_2}^i$ which is formally expressed by $(O_{k_1}^f \wedge O_{k_2}^i)$. The information concerning the resource and related operations and discrete locations are saved as the attributes of the shape and will be used in the following steps.

Algorithm 1 Collision detection

Input: $\mathcal{R}$ – the set of resources.
Output: $\mathcal{X}$ – the set of pairs of colliding shapes;
$\mathcal{S}$ – the set of generated shapes.

foreach $R_\ell \in \mathcal{R}$ **do**
  foreach $O_k \in R_\ell$.*Operations()* **do**
    $(S_k^i, S_k^e, S_k^f) \leftarrow$ *CreateShapeByLocation*(O_k^i, O_k^e, O_k^f)
    $\mathcal{S} \leftarrow (S_k^i, S_k^e, S_k^f)$
  end for
end for
while $\mathcal{S} \neq \emptyset$ **do**
  $S_i = \mathcal{S}$.*RemoveFirst()*
  foreach $S_j \in \mathcal{S}$ **do**
    if $S_i.Resource() \neq S_j.Resource()$ **then**
      if S_i *intersects with* S_j **then**
        $\mathcal{X} \leftarrow (S_i, S_j)$
      end if
    end if
  end for
end while
return $\mathcal{X}, \mathcal{S}$

Example 1 (Continue): Algorithm 1 is employed to identify the collisions in the example cell. The input of the algorithm is the set $\mathcal{R} = \{Rb1, Rb2, C\}$. In the first iteration, the shapes for all operations, e.g. $\{S_1^i, S_1^e, S_1^f\}$ for operation O_1, are created and added to the set $\mathcal{S}$ by simulating the operations in Dassault Systèmes DELMIA V5. For the operation locations $\{O_1^f, O_2^i\}$ and $\{O_3^f, O_4^i\}$ of the resources $Rb1$ and $Rb2$, which are in sequence, one shape for each resource is created, $S_{1,2}^{fi}$ and $S_{3,4}^{fi}$ respectively, which are represented by the combined discrete locations $(O_1^f \wedge O_2^i)$ and $(O_3^f \wedge O_4^i)$.

TABLE I
PAIRS OF COLLIDED SHAPES IN THE SET OF $\mathcal{X}$

Shape S_i	Shape S_j
S_1^e	S_5^e
$S_{1,2}^{fi}$	S_4^e
S_2^e	$S_{3,4}^{fi}$

In the next step, the pairwise intersection test is applied on the set $\mathcal{S}$, and the pairs of colliding shapes is added to the set $\mathcal{X}$, see Table I. Fig. 3 illustrates the pair of colliding shapes (S_1^e, S_5^e), which indicates the collision between $Rb1$ while executing the operation O_1, and C while executing the operation O_5. □

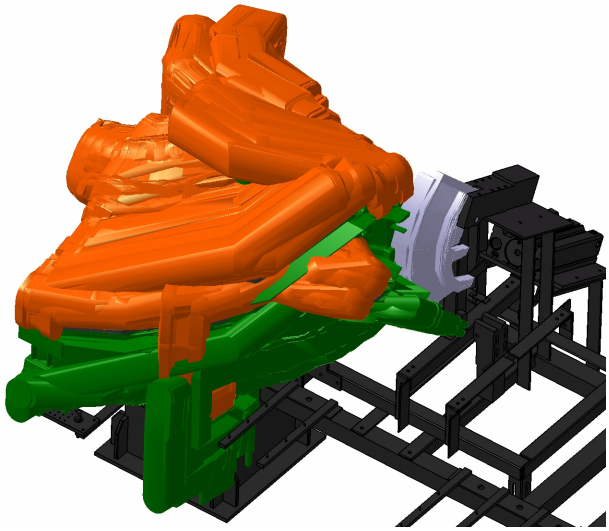

Fig. 3. The shapes S_1^e and S_5^e which are representing the collision between the resources $Rb1$ and C in their execution location to perform operations O_1 and O_5 respectively.

C. Interlocks generation

For each pair of colliding shapes, the corresponding forbidden locations are identified. In order to avoid collision, reaching the forbidden locations are eliminated by adding guards to the corresponding events of the two involved operation models, see Algorithm 2. The input of the algorithm is the set of pairs of colliding shapes $\mathcal{X}$.

Algorithm 2 Guard generation on transition conditions

Input: $\mathcal{X}$ – the set of pairs of colliding shapes;

while $\mathcal{X} \neq \emptyset$ **do**
  $(S_1, S_2) \leftarrow \mathcal{X}$.*RemoveFirstPair()*
  $(\mathcal{O}_1, \mathcal{O}_2) \leftarrow$ *DiscreteLocationSetFromShape*(S_1, S_2)
  for $z = 1$ **to** 2 **do**
    $O_{t_1}^{\ell_1} = \mathcal{O}_z$.*RemoveFirst()*
    $k \leftarrow t_1$
    if $(\exists O_{t_2}^{\ell_2} \in \mathcal{O}_z : \ell_2 = f)$ **then**
      $k \leftarrow t_2$
    end if
    $C_k^{\uparrow} = C_k^{\uparrow} \wedge \neg \bigwedge \mathcal{O}_{3-z}$
    if $(\exists O_m^y \in \mathcal{O}_{3-z} : y = f)$ **then**
      $C_k^{\uparrow} = C_k^{\uparrow} \wedge \neg O_m^e$
    end if
  end for
end while

In the first step of the algorithm, one pair of colliding shapes, (S_1, S_2), is selected (and removed) from the set $\mathcal{X}$. For each of the colliding shapes, S_1 and S_2, the corresponding operation locations are retrieved from the shapes. Since each shape represents a discrete location (or combination of discrete locations) in the related operation(s), to avoid the collision, it is enough to forbid operations to enter the collision location concurrently. This condition is expressed by adding the safety guards to the transition condition $C_k^{\uparrow}$ of operations. If the discrete location O_k^f should be forbidden, since the event $O_k^{\downarrow}$ is an uncontrollable event, the discrete location O_k^e is also forbidden to avoid an uncontrollable supervisor.

Example 1 (Continue): The input of the Algorithm 2 is the set of pairs of colliding shapes $\mathcal{X}$ from previous step. In the first iteration, the pair of (S_1^e, S_5^e) is taken. The function *DiscreteLocationSetFromShape()* retrieves the location sets, $\mathcal{O}_1 = \{O_1^e\}$ and $\mathcal{O}_2 = \{O_5^e\}$. In the next step, the condition $\neg O_5^e$, will be added to the transition condition $C_1^{\uparrow}$, and the condition $\neg O_1^e$ to the $C_5^{\uparrow}$, respectively. These conditions avoid the robot and the clamp to be in their execution location concurrently. In the next iteration another pair is taken, and with the same procedure, necessary conditions will be added to their operation models. Table II, shows the generated conditions for all operation models. □

TABLE II
GENERATED CONDITIONS FROM ALGORITHM 2

Transition condition	Added boolean condition
$C_1^{\uparrow}$	$\neg O_4^e \wedge \neg O_5^e$
$C_2^{\uparrow}$	$\neg(O_3^f \wedge O_4^i) \wedge \neg O_3^e$
$C_3^{\uparrow}$	$\neg O_2^e$
$C_4^{\uparrow}$	$\neg(O_1^f \wedge O_2^i) \wedge \neg O_1^e$
$C_5^{\uparrow}$	$\neg O_1^e$

D. User specification

Without any restrictions on the individual operations the basic assumption is that all operations can be executed in parallel [7]. On the other hand, in a manufacturing system there are precedence relations between operations. Restrictions on the order between different operations are formally expressed by logical *preconditions on individual operations* and indicated by guards on corresponding operation models. Since the operations are guaranteed to be collision-free, the user can freely specify desired SOPs without considering collision problems.

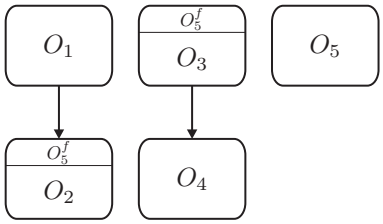

Fig. 4. Desired sequences where preconditions are added such that operation O_5 needs to be finished before O_2 and O_3 are allowed to start.

Example 1 (Continue): Fig 4 shows a user specification by ordering the operation O_5 to precede the operations O_2 and O_3. Hence, the transition conditions for O_2 and O_3 become $C_2^{\uparrow} = C_2^{\uparrow} \wedge O_5^f$ and $C_3^{\uparrow} = C_3^{\uparrow} \wedge O_5^f$, respectively. □

E. Synthesis

So far, the guards that have been generated and attached to the operation models, i.e. EFAs, yield a collision-free system that fulfills the user's specification, e.g. the sequence of operations. Furthermore, it is necessary to synthesize a non-blocking, controllable and maximally permissive supervisor [8]. Traditionally, the synthesis procedure for EFAs was carried out by first flattening the EFAs to ordinary finite automata, and then perform a monolithic synthesis. In this way, the states of the final supervisor are represented explicitly, which has some main drawbacks when the supervisor becomes very large in terms of the number of states:

- Converting EFA to ordinary automata may be time-consuming.
- The supervisor may be untraceable for the users and hard to understand.
- It may not be possible to implement the controller based on the supervisor on a PLC with limited memory.
- Since the number of states increases exponentially during the monolithic synthesis, *state space explosion* may occur, and thus, no supervisor will be computed.

To overcome the above problems, we perform the synthesis based on the approach proposed in [10]. In this framework, the synthesis is performed directly on the EFAs, rather than flattening them to ordinary automata. The result will be a non-blocking, controllable, and maximally permissive monolithic supervisor.

Based on the monolithic supervisor, a set of reduced guards are generated and attached to the original EFAs, representing the conditions that should hold in order to guarantee that the system preserves the supervisor's properties. In Section IV, this approach is applied to a very large system, where a huge supervisor is represented by a few number of small guards.

Example 1 (Continue): With all collision-free operation models including the user specification, the supervisor is generated in the Supremica software. Fig. 5 shows the synchronized model, in which the cross marks are the blocking states (removed by synthesis) and slash marks are the forbidden states (removed by the collision avoidance specification). After removing the blocking and uncontrollable states, the supervisor will be the sequence of operations $(O_1 \rightarrow O_5 \rightarrow O_2 \rightarrow O_3 \rightarrow O_4)$, $(O_5 \rightarrow O_1 \rightarrow O_2 \rightarrow O_3 \rightarrow O_4)$ or $(O_5 \rightarrow O_3 \rightarrow O_4 \rightarrow O_1 \rightarrow O_2)$.

Also, it is possible to generate the EFA model of the supervisor. Fig. 6 shows the EFA supervisor model. It is interesting that few guards is enough to generate the non-blocking supervisor. □

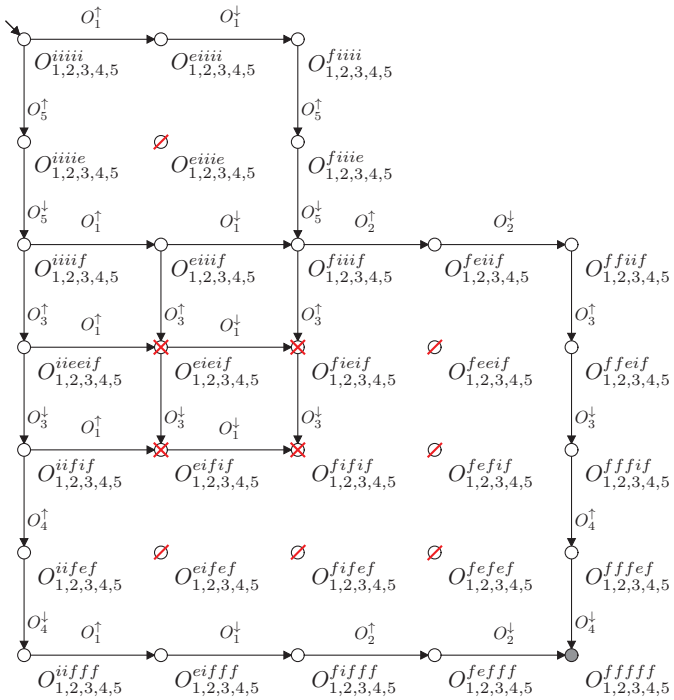

Fig. 5. The monolithic synchronized model of the example cell. The cross marks indicate the blocking states and the slash marks indicates the forbidden states. The notation $O^{eiiii}_{1,2,3,4,5}$ means the combined operation locations $O^e_1, O^i_2, O^i_3, O^i_4, O^i_5$.

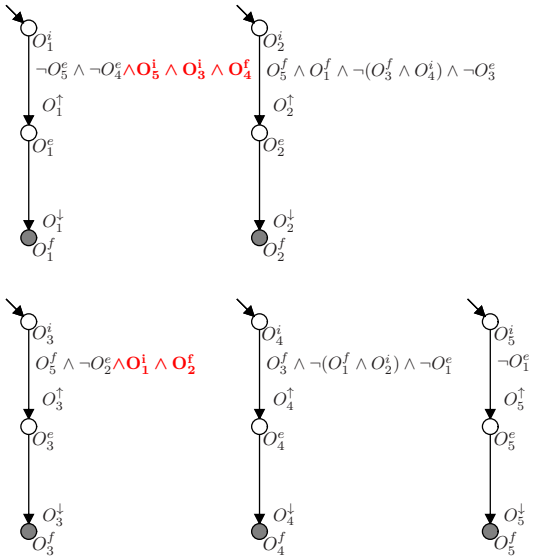

Fig. 6. The supervisor for the example cell using direct synthesis on EFAs. The bold red guards are added by the synthesis algorithm to remove the blocking states.

IV. Case Study

The method described above has been implemented using three existing software, Dassault Systèmes DELMIA V5, as a 3D simulation environment, for generating the operation models, collision detection and interlocks generation, Sequence Planner [16], to add the desired specification to the operations, and Supremica, for the verification and synthesis. The program is used to automatically generate the controller for a robot cell at Chalmers University of Technology (PPU Lab). The cell consists of seven resources: two ABB robots, a fixture with four clamps, and a conveyor. The desired behaviour for the cell is the following. Two parts are loaded by operator and carried to the work station on the conveyor.

First, robot $Rb1$ picks a part and places it on the fixture, and then robot $Rb2$ takes the second part and places it on the fixture. Afterwards, the clamps are closed to fixate the parts. Then, the robots start to assemble the product by drilling and pop-riveting the parts. Finally, the clamps are opened and the finished product is ready to be unloaded.

The input to the method is the 3D simulation model of the cell including predefined operations for the resources in Dassault Systèmes DELMIA V5. The generated operation models with collision avoidance guards are structured and exported to a XML-format. In Sequence Planner, from the XML-format, the operation models are imported. Here, desired SOPs are visually created as the user specification. Finally, the controllable and non-blocking supervisor is generated in Supremica based on the operation models including the desired user specifications.

To test the capability of the synthesis approach based on guard generation, the cell described above is duplicated and run concurrently. After synthesis, the number of reachable states for the closed loop system and the generated supervisor are given in Table III, including some more relevant cell information.

TABLE III
STATICS OF THE SUPERVISOR FOR TWO CELLS

Complex Case Study	Numbers
No. of Resources	14
No. of Operation	68
No. of Pairs of colliding shapes	84
No. of Generated Guards to avoid collisions	164
No. of Closed-loop states	1.35×10^{11}
No. of Supervisor states	2.15×10^{9}
No. of generated guards based on the SCT synthesis	4
Time to compute using BDD	3.2 seconds

V. CONCLUSIONS AND FUTURE WORK

A method for automatic generation of a non-blocking controller that generate collision-free flexible manufacturing cells has been presented in this paper. Two crucial steps are included, first a safety specification is generated automatically based on 3D simulation and formal operation models. A collision-free system is achieved by considering pairs of colliding shapes as forbidden states. Secondly, a non-blocking and controllable supervisor is generated based on guard generation. The guards are computed by binary decision diagrams. A case study shows that large systems can be handled including billions of reachable states and even more, still only resulting in a few guards that are easily to interpret and implement in for instance a PLC or a robot controller.

Future work will include cycle time and/or energy optimization based on MILP and MINLP solvers. A stronger interaction between the 3D simulation software and our suggested synthesis method will also further improve the usability of the suggested methodology.

REFERENCES

[1] Dassault Systèmes DELMIA V5 and DELMIA Automation V5, 2010, http://www.3ds.com.

[2] ABB RobotStudio, 2010, http://www.robotstudio.com.

[3] Siemens Tecnomatix, 2010, http://www.tecnomatix.com.

[4] E. W. Endsley, E. E. Almeida, and D. M. Tilbury, "Modular finite state machines: Development and application to reconfigurable manufacturing cell controller generation," *Control Engineering Practice*, vol. 14, no. 10, pp. 1127–1142, 2006.

[5] K. Andersson, J. Richardsson, B. Lennartson, and M. Fabian, "Coordination of Operations by Relation Extraction for Manufacturing Cell Controllers," *IEEE Transactions on Control Systems Technology*, pp. 1–17, 2009.

[6] K. Bengtsson, B. Lennartson, Y. Chengyin, P. Falkman, and S. Biller, "Operation-oriented specification for integrated control logic development," in *2009 IEEE International Conference on Automation Science and Engineering*. IEEE, 2009, pp. 183–190.

[7] B. Lennartson, K. Bengtsson, C. Yuan, K. Andersson, M. Fabian, P. Falkman, and K. Akesson, "Sequence Planning for Integrated Product, Process and Automation Design," *Conditionally accepted for IEEE Transactions on Automation Science and Engineering*, 2010.

[8] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[9] M. Skoldstam, K. Akesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *IEEE Conference on Decision and Control*, 2007, pp. 3387–3392.

[10] S. Miremadi, K. Akesson, and B. Lennartson, "Symbolic Supervisory Synthesis on Extended Finite Automata," in *submitted to IEEE Transactions on Automation Science and Engineering*, 2010.

[11] A. Vahidi, B. Lennartson, and M. Fabian, "Efficient Analysis of Large Discrete-Event Systems with Binary Decision Diagrams," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, 2005, pp. 2751–2756.

[12] K. Akesson, M. Fabian, H. Flordal, and R. Malik, "Supremica - An integrated environment for verification, synthesis and simulation of discrete event systems," in *2006 8th International Workshop on Discrete Event Systems*. IEEE, 2006, pp. 384–385.

[13] C. A. R. Hoare, "Communicating Sequential Processes," *Communications of the ACM*, vol. 21, pp. 666–677, 1985.

[14] P. Jiménez, F. Thomas, and C. Torras, "3D Collision Detection: A Survey," *Computers and Graphics*, vol. 25, pp. 269–285, 2000.

[15] M. C. Lin and S. Gottschalk, "Collision Detection Between Geometric Models: A Survey," in *In Proc. of IMA Conference on Mathematics of Surfaces*, 1998, pp. 37–56.

[16] E. Ohlson and C. Torstensson, "Development, implementation and testing of Sequence Planner - A concept for modeling of automation systems," Tech. Rep. EX/2009, Chalmers University of Technology, 2009.